**Data Ingestion API System - Documentation**

**1. Overview**

This project implements a priority-based asynchronous **Data Ingestion API System** that accepts a list of IDs and processes them in batches, respecting a strict rate limit and honoring priority levels (HIGH, MEDIUM, LOW). It consists of two RESTful API endpoints to submit ingestion requests and retrieve their processing status.

**2. Tech Stack Used**

* **Backend**: Node.js + Express.js
* **Queueing/Async Handling**: Custom JavaScript Logic + setTimeout
* **Data Store**: In-Memory JavaScript Objects
* **Deployment**: Render / Cyclic (Node backend hosting)
* **Testing**: Postman
* **Version Control**: GitHub

**3. API Endpoints**

**POST /ingest**

* **Description**: Submit a new ingestion job with a list of IDs and priority.
* **Request Payload:**

{

"ids": [1, 2, 3, 4, 5],

"priority": "HIGH"

}

* **Response Example:**

{

"ingestion\_id": "abc123"

}

* **Behavior:**
  + Splits IDs into batches of maximum 3
  + Each batch is enqueued with delay (1 batch per 5 seconds)
  + Priority dictates execution order (HIGH > MEDIUM > LOW)

**GET /status/:ingestion\_id**

* **Description**: Returns the processing status of an ingestion request.
* **Response Example:**

{

"ingestion\_id": "abc123",

"status": "triggered",

"batches": [

{

"batch\_id": "uuid-1",

"ids": [1, 2, 3],

"status": "completed"

},

{

"batch\_id": "uuid-2",

"ids": [4, 5],

"status": "triggered"

}

]

}

* **Batch Status Options**: yet\_to\_start, triggered, completed
* **Overall Status Logic:**
  + All batches yet\_to\_start → yet\_to\_start
  + At least one triggered → triggered
  + All completed → completed

**4. Processing Logic**

1. Client sends ingestion request with IDs and priority.
2. Server breaks IDs into batches (3 IDs max).
3. Batches are enqueued into a global queue sorted by priority and submission time.
4. Background processor picks the highest priority batch every 5 seconds.
5. Each ID simulates data fetching from a mock external source with a delay.
6. Status updates are stored in an in-memory store.

**5. Sample Test Scenarios**

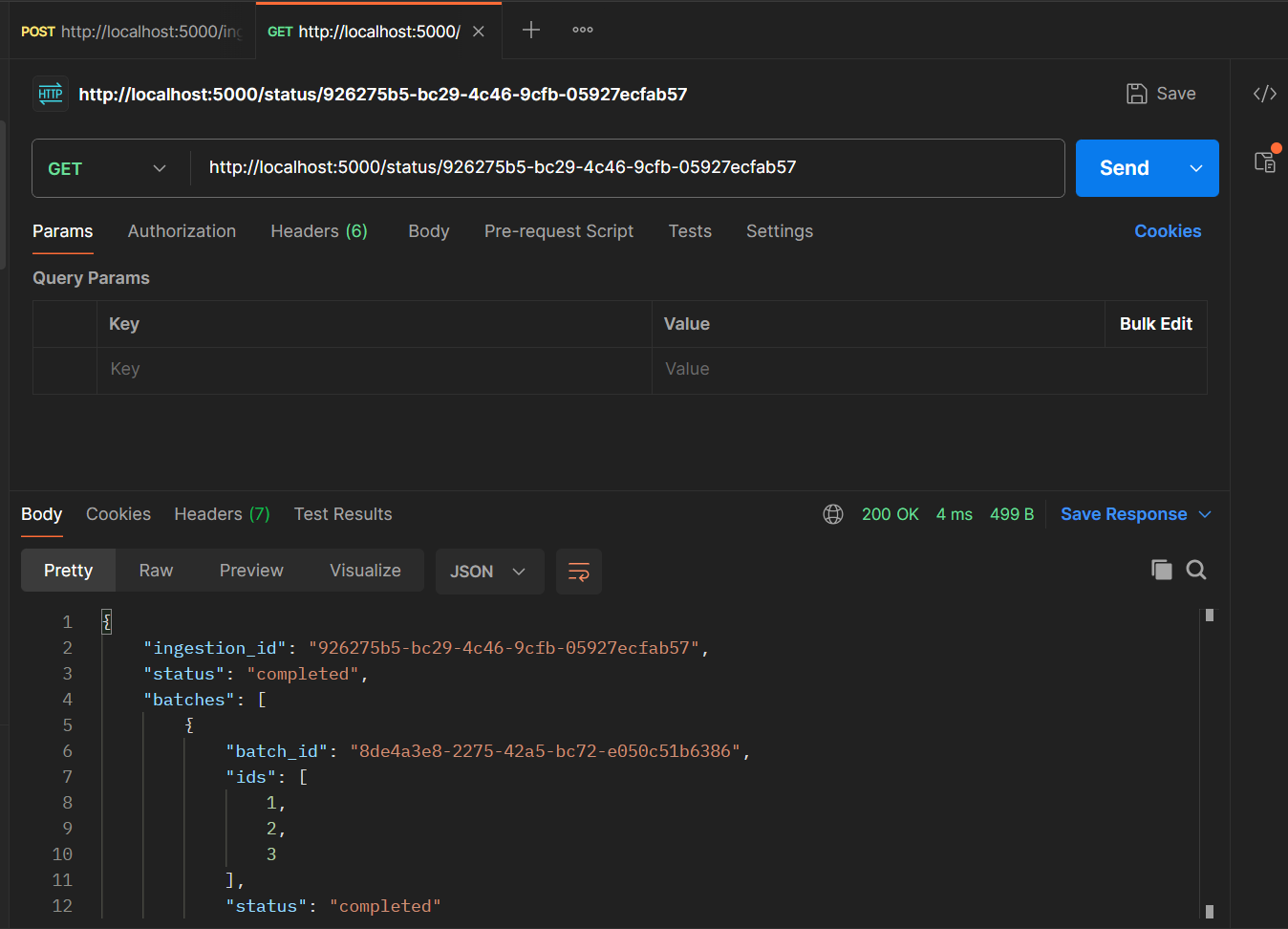
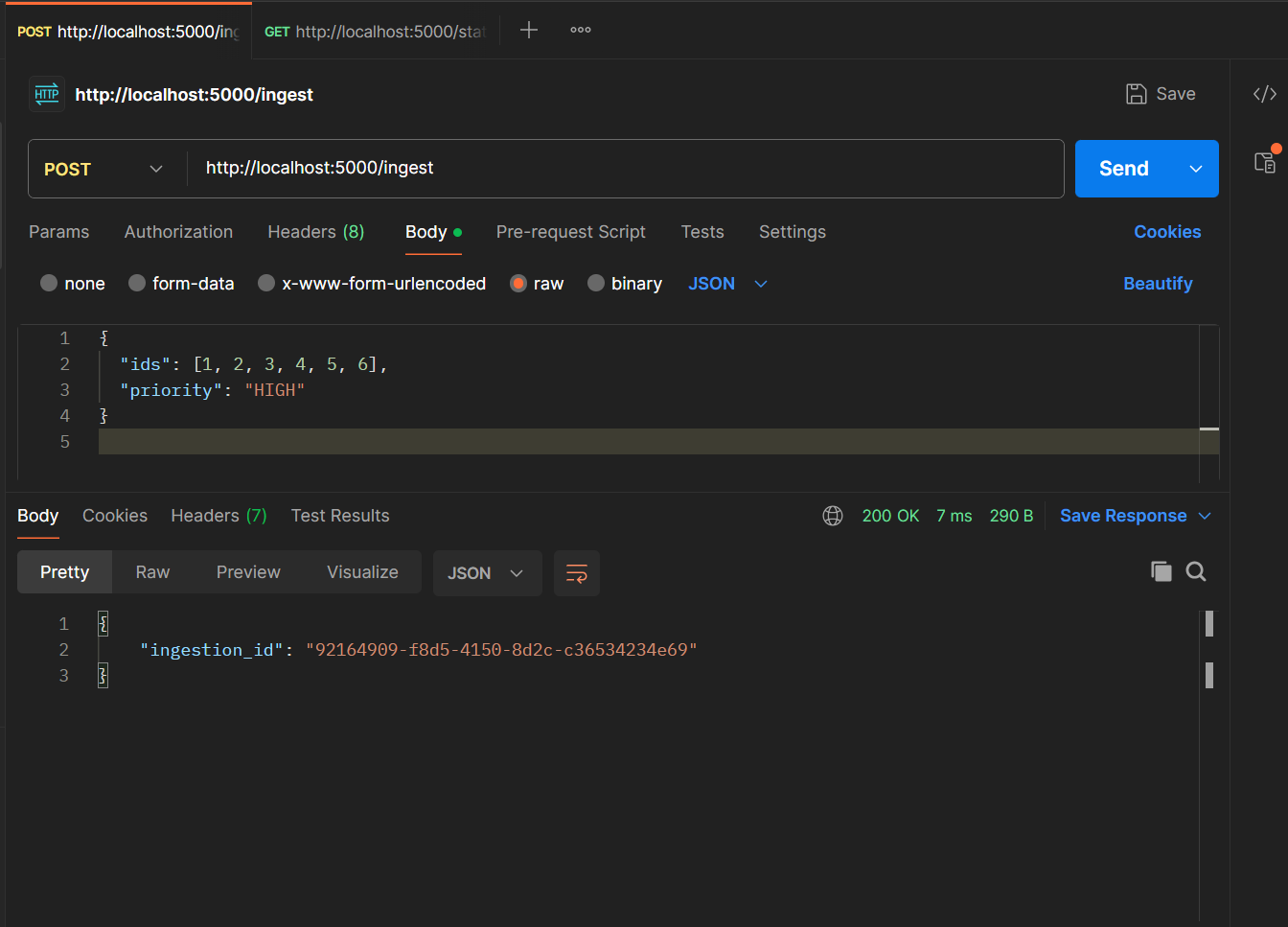
* Request 1 (T0): { ids: [1,2,3,4,5], priority: MEDIUM }
* Request 2 (T4): { ids: [6,7,8,9], priority: HIGH }

**Processing Order:**

* T0-T5: 1, 2, 3 (MEDIUM)
* T5-T10: 6, 7, 8 (HIGH)
* T10-T15: 9, 4, 5 (HIGH + leftover MEDIUM)

**6. Screenshots**

*Include the following screenshots:*



**7. Known Limitations**

* In-memory data store: Loses all state on restart
* No retry mechanism for failed tasks
* No database or persistent queuing system

**8. Future Improvements**

* Use Redis for queue and data persistence
* Add retry and timeout handling
* Use cron jobs or job schedulers
* Add rate limiting library for production robustness

**9. Deployment & GitHub**

* **Hosted URL**: https://data-ingestion-api-16mg.onrender.com
* **GitHub Repo**: <https://github.com/AnshArora07/data-ingestion-api>

**10. Author**

**Name**: Ansh Arora